CSE351 Autumn 2012 – Final Exam (12 Dec 2012)

Please read through the entire examination first! We designed this exam so that it can be completed in 90 minutes and, hopefully, this estimate will prove to be reasonable.

There are 5 problems for a total of 200 points. The point value of each problem is indicated in the table below and at every part of every problem. Write your answer neatly in the spaces provided. If you need more space (you shouldn't), you can write on the back of the sheet where the question is posed, but please make sure that you indicate clearly the problem to which the comments apply. Do NOT use any other paper to hand in your answers. If you have difficulty with part of a problem, move on to the next one. They are independent of each other.

The exam is CLOSED book and CLOSED notes. Please do not ask or provide anything to anyone else in the class during the exam. Make sure to ask clarification questions early so that both you and the others may benefit as much as possible from the answers.

Name: _____

ID#: _____

Problem	Max Score	Score
1 (Appetizer)	25	
2 (Soup/Salad)	35	
3 (Entrée 1)	60	
4 (Entrée 2)	40	
5 (Dessert)	40	
TOTAL	**200**	

1. Appetizer (Short Answers) – 25pts total (5pts each)

A. What is cached in the data portion of a TLB (translation-lookaside buffer) line in a virtual memory system? (check all that apply)

- ☐ A page location in physical memory
- ☐ A page location in virtual memory
- ☐ A page location on disk
- ☐ None of the above

B. Why are virtual memory pages so much bigger than memory cache blocks? (check one)

- ☐ There is no special reason – that is just how it is
- ☐ Getting a virtual memory from disk is slow – get more data to save time
- ☐ Memory cache blocks are smaller because we only access one word at a time

C. When we need more memory on the dynamic heap, why not just double the heap size each time? (check all that apply)

- ☐ It would lead to low memory utilization
- ☐ The heap may be constrained by other elements in memory such as the stack
- ☐ Virtual memory may not have enough pages
- ☐ It will lead to worse cache performance

D. Which of the following is guaranteed to lead to a virtual memory page fault? (check all that apply)

- ☐ A TLB miss
- ☐ A page table entry pointing to a physical memory location
- ☐ A page table entry pointing to a disk location

E. When starting a new process via a fork, which of the following happens? (check all that apply)

- ☐ Child process is given an exact copy of the parent memory when it starts
- ☐ The TLB is pre-filled with the page table entries for the child process
- ☐ The physical memory cache is flushed so it starts fresh for the child process

2. Soup or Salad (Cache Memory) – 35 pts total (5/A, 5/B, 15/C, 10/D)

Suppose we have a system with the following properties:

- Memory accesses are to 4-byte words
- Addresses are 12 bits wide
- The cache is two-way set associative, with a 8-byte block size and 4 sets
- A cache hit has 10ns latency and a cache miss has 100ns latency

A. What is the size of the cache in bytes? _____

B. Fill in which bits are used for the byte offset (CO), set index (CI), and cache tag (CT)

C. The following code is run on this machine:

```
int arr[4][4], i, j;
for (i = 0; i < 4; i++) {
   for (j = 0; j < 4; j++) {
      arr[j][i] = i * j;
   }
}
```

Assume the following:

- `i` and `j` are stored in registers
- `sizeof(int)` = 4 bytes
- `arr` is stored in row-major order starting at 0x00000000 in memory
- The cache is initially empty

Number of cache hits: _____

Number of cache misses: _____

Total latency: _____

D. The program above iterates through the array columns rather than the rows. If `arr[j][i]` is changed to `arr[i][j]` so that it iterates through the rows, will there be a performance increase? If so, how much?

3. Entrée 1 (Heap Data Structures) – 60pts total (10/A, 20/B, 10/C, 20/D)

Address .	Data .
0x a188	abababab abababab
0x a180	abababab abababab
0x a178	abababab abababab
0x a170	abababab abababab
0x a168	abababab abababab
0x a160	abababab abababab
0x a158	abababab abababab
0x a150	41000000 00000000
0x a148	42000000 00000000
0x a140	abababab abababab
0x a138	abababab abababab
0x a130	abababab abababab
0x a128	abababab abababab
0x a120	20a00000 00000000
0x a118	90a10000 00000000
0x a110	42000000 00000000
0x a108	abababab abababab
0x a100	abababab abababab
0x a0f8	abababab abababab
0x a0f0	abababab abababab
0x a0e8	abababab abababab
0x a0e0	33000000 00000000
0x a0d8	abababab abababab
0x a0d0	abababab abababab
0x a0c8	abababab abababab
0x a0c0	21000000 00000000
0x a0b8	a2000000 00000000
0x a0b0	abababab abababab
0x a0a8	abababab abababab
0x a0a0	abababab abababab
0x a098	abababab abababab
0x a090	abababab abababab
0x a088	abababab abababab
0x a080	abababab abababab
0x a078	abababab abababab
0x a070	abababab abababab
0x a068	abababab abababab
0x a060	abababab abababab
0x a058	abababab abababab
0x a050	abababab abababab
0x a048	abababab abababab
0x a040	abababab abababab
0x a038	abababab abababab
0x a030	10990000 00000000
0x a028	10a10000 00000000
0x a020	a2000000 00000000
0x a018	23000000 00000000
0x a010	10990000 00000000
0x a008	20a00000 00000000
0x a000	23000000 00000000

After a hard night of 351 studying, you dream that you are stuck in a malicious ***little-endian 64-bit*** computer system. You run to escape via the nearest I/O port, fighting off guards in a light cycle race. You dig into the code that controls the port and find part of the heap for the currently running program. If you change the right bits, you can allocate enough heap space to run a password cracking subroutine and escape from your terrible dream. However, if you corrupt the heap, the program may crash, trapping you inside forever!

You may draw/mark/use the heap memory dump (on the left) as you wish. You can assume an ***explicit free list*** as in your last assignment. Note that memory contents are little-endian.

The used-flags are also the same as the assignment; in the size of a block, the ***2^0 bit*** is set if the current block is allocated and the ***2^1 bit*** is set if the previous block is allocated.

You will need this memory map for reference as you work out the problem on the next page.

Recall that blocks are defined as follows:

```
struct BlockInfo {
      size_t sizeAndTags;
      BlockInfo* next;
      BlockInfo* prev;
}
```

A. You notice that a call to the malloc routine just returned the address **0xa008**. Describe the block that was just allocated:

Start address of payload: 0x ___a008__

Start address of block: 0x _____

Size of block: _____ bytes (in decimal)

B. The dump of memory shown on the previous page is just after that block was allocated (free list pointers have yet to be overwritten). Describe the free list, starting from the block that was just allocated:

Start of next free block: 0x _____

Size of next free block: _____ bytes (in decimal)

Start of next next free block: 0x _____

Size of next next free block: _____ bytes (in decimal)

C. Your password-cracking subroutine needs 144 bytes of heap space. Assume no other malloc() calls are made while you modified the memory. Find a free block that is large enough to hold the 144 bytes. What are some properties of this block?

Address of the block: 0x _____

Size of the block: _____ bytes (in decimal)

Bytes of internal fragmentation: _____ bytes (in decimal)

D. What steps do you need take to allocate this block for your escape? Do not worry about splitting the block and placing the part you don't need on the free list. However, please state the reason for each step right above the address and value. You are given Step 1 as an example. Write new values as they would appear in the memory map at the start of this problem – in little-endian. You are given room for up to 6 steps, but you may NOT need all of them – just use as many as needed.

Address to change (hex). Value to write to that address (hex, little-endian).

Step 1: Change the block's header to mark it allocated .

0x ____a020____ 0x __a3000000__

Step 2: .

0x _____ 0x _____

Step 3: .

0x _____ 0x _____

Step 4: .

0x _____ 0x _____

Step 5: .

0x _____ 0x _____

Step 6: .

0x _____ 0x _____

4. Entrée 2 (Virtual Memory) – 40pts total (15/A, 10/B, 5/C, 5/D, 5/E)

Answer the following questions with a couple of sentences.

A. List two reasons why we have virtual memory in modern computer systems.

B. What is indirection and what role does it have in allowing programs to refer to virtual memory instead of physical memory?

C. When does a TLB (Translation Lookaside Buffer) miss occur? What is the difference between a TLB miss and a page fault?

D. In a virtual memory system with 64-bit addresses and with 256T page table entries ($1T = 2^{40}$), how big are memory pages (in bytes)?

E. If for the same system as in (D), physical memory has 32-bit physical addresses, determine the number of bits in the VPN (virtual page number), VPO (virtual page offset), PPN (physical page number), and PPO (physical page offset).

5. Dessert (Pointers in C) – 40pts total (A/10, B/10, C/10, D/10)

You are given the following definitions for some graphics code:

```
struct pixel {
     char r;
     char g;
     char b;
};

pixel buffer[480][640];

struct color {
     char r;
     char g;
     char b;
}
```

The following code snippets appear in the program you have been given to analyze. Write a comment block for each one of these C functions describing its parameters and return value as well as each assignment and type cast used in the body.

```
pixel *getPixel (int i, int j) {
     return &buffer[i][j];
}
```

```
void changeBufferToColorOfPixel(pixel* chosenPixel) {
     color newcolor = (color) *chosenPixel;
     changeToColor(&newcolor);
}
```

```
void changeToColor (color* newcolor) {
     pixel newpixel = (pixel) *newcolor;
     int i, j;
     for (i = 0; i<480; i++) {
          for (j = 0; j < 640; j++) {
               buffer[i][j] = newpixel;
          }
     }
}
```

A. getPixel

B. changeBufferToColorOfPixel

C. changeToColor

D. In their respective functions, where are newcolor and newpixel allocated in memory?